CS 202 assignment 1

Tejas R-201050, S Pradeep-200826

January 2022

1 Sudoku Pair Solver

We are given a k-dimension Sudoku, and our problem is to fill in the empty cells, following the Sudoku constraints. We try to do so using propositional logic.

Implementation: We need to think about a set of propositional variables, such that if we are able to find if they are true or false, we could extract a solution for a given Sudoku from them. Similar to the graph colouring problem, where we chose propositional variables of the form x_{ab}, to denote that the a^{th} vertex is assigned the b^{th} colour, here we choose variables of the form x_{rcv}, which denotes that the cell $[r, c]$ (i.e the cell in row a and column b) is assigned the number c. The number c may vary from 1 *to* k^2. Since we have k^4 cells, and each cell can be assigned one of k^2 numbers, we get a total of k^6 atomic propositions for a k-dimension Sudoku- for example, when k=3, the Sudoku has 81 cells and each cell can be assigned a number from 1-9 - hence we have 3^6 atomic propositions. Each cell (r,c) containing value v, denoted by atomic proposition x_{rcv}, is mapped to a natural number n by a function f given by

$$n = f(r, c, v) = (r-1) * k^2 * k^2 + (c-1) * k^2 + (v-1) + 1 \tag{1}$$

We use n to denote that the cell at (r,c) is filled with value v.

Now we try to convert the constraints of a Sudoku into logical statements- 1) One cell can only be assigned one number - so that means for a given doublet (r, c), only one of $x_{rc1}, x_{rc2}, ...x_{rck^2}$ is true. This also implies that at-most one of them and at-least one of them is true. Hence, $\forall\ (r, c)\ , \neg(x_{rci} \wedge x_{rcj})$. Since at-least one of them must be true, $\forall\ (r, c)\ , (x_{rc1} \vee x_{rc2} \vee ...x_{rck^2})$. Since each digit occurs at-least once in one row, the corresponding logical statement can be thought of as $\forall\ (r, v)$, $(x_{r1v} \vee x_{r2v} \vee x_{r3v} \vee ...x_{rk^2v})$. Since each digit occurs at-least once in one column, the corresponding logical statement can be thought of as $\forall\ (c, v)$, $(x_{1cv} \vee x_{2cv} \vee x_{3cv} \vee ...x_{k^2cv})$. Since each digit occurs at-least once in each sub-grid. An example of a logical statement corresponding to the first sub-grid in a k dimensional sudoku is $\forall\ v, (x_{11v} \vee x_{12v} \vee ... \ x_{1kv} \vee x_{21v} \vee x_{22v} \vee ... \ x_{kkv})$. This follows similarly for other subgrids in the k dimensional sudoku. Having an atleast statement for each digit in each row, column and subgrid ensures that exactly one digit is there in each row, column and subgrid. This is because of

the restriction on each cell having exactly one element and the fact that there are k^2 numbers to fill and k^2 cells exactly. The same set of clauses hold for the second sudoku also, except the row numbers(r for the second sudoku start from $k^2 + 1$ and go till $2k^2$. Additionally, the numbers in the corresponding cells in the two sudokus must not be same. This can be achieved by the logical statement: $\forall\ (r, c, v),\ r \in [1, k^2],\ \neg(x_{rcv} \wedge x_{r+Ncv})$. All of these clauses solved together provide us the required model for the solved sudoku. Using this model, we print out the solved sudoku.

Assumptions: We assume that the given csv file contains the input in the correct format. We also assume that the k value given in the terminal is consistent with the dimension of the sudokus given as input in the csv file. The output will show $2k^2$ rows of numbers which represent the solved sudoku.

2 Sudoku Pair Generator

To write a k-sudoku puzzle pair generator. The puzzle pair must be maximal (have the largest number of holes possible) and must have a unique solution.

Implementation: We first generate two solved sudokus, using two nearly empty sudokus, such that they are a sudoku pair(no corresponding value is the same). (To make the suduko pair generation random, we fill a random cell with a random value , and then proceed to find the solved sudoku pair).

Then we used backtracking to find the maximal sudoku pair. Each cell (r,c) containing value v, denoted by atomic proposition x_{rcv}, is mapped to a natural number n by a function f given by

$$n = f(r, c, v) = (r - 1) * k^2 * k^2 + (c - 1) * k^2 + (v - 1) + 1 \tag{2}$$

We use n to denote that the cell at (r,c) is filled with value v.

The set of all atomic propositions, which correspond to cells which are filled in the given pair of sudokus, at any point of time, is called pos(). Initially pos() has $2 * k^4$ elements, corresponding to the filled sudoku pair, since each of the $2 * k^4$ cells has exactly one value $<= k^2$ filled in it.

In each iteration of the recursion, we iterate over the elements of pos. At each element t in pos, we pop t from pos, where t corresponds to cell (a,b) with value v(say). We consider the sudoku pair with value in cell (a,b) set to 0. We initialise solver g1() with the appropriate clauses - g1() has all clauses to ensure standard sudoku constraints, and the pair constraint, and we add all the current elements in pos() in every recursive iteration. We use g1 to solve for the modified sudoku pair. If the number of models we get for g1() is greater than one, then that means we must restore t to the pos,and(hence automatically) v to the cell (a,b) since the sudoku pair will have non unique solution on deleting t from pos(). If on deleting t from pos(), g1() is uniquely satisfiable, then we need not restore it to pos() and we continue iterating along its elements. The base case for the recursion, is when pos() has no more elements which can be removed from it without violating the uniqueness of the solution to the puzzle

pair. This corresponds to the case when we iterate over all the elements of pos() in the final recursive call, and for each element g1() gives multiple models, and hence we append each element back into pos(). So when in a recursive call, the number of elements in pos remains constant, we can end the recursion and return pos(). We then use the elements in pos() to fill in the sudoku pair, by finding (r,c,v) for every element for the pos which corresponds to the maximal sudoku pair.

Assumptions: We assume the dimension k of the sudoku will be given as input on the command line.